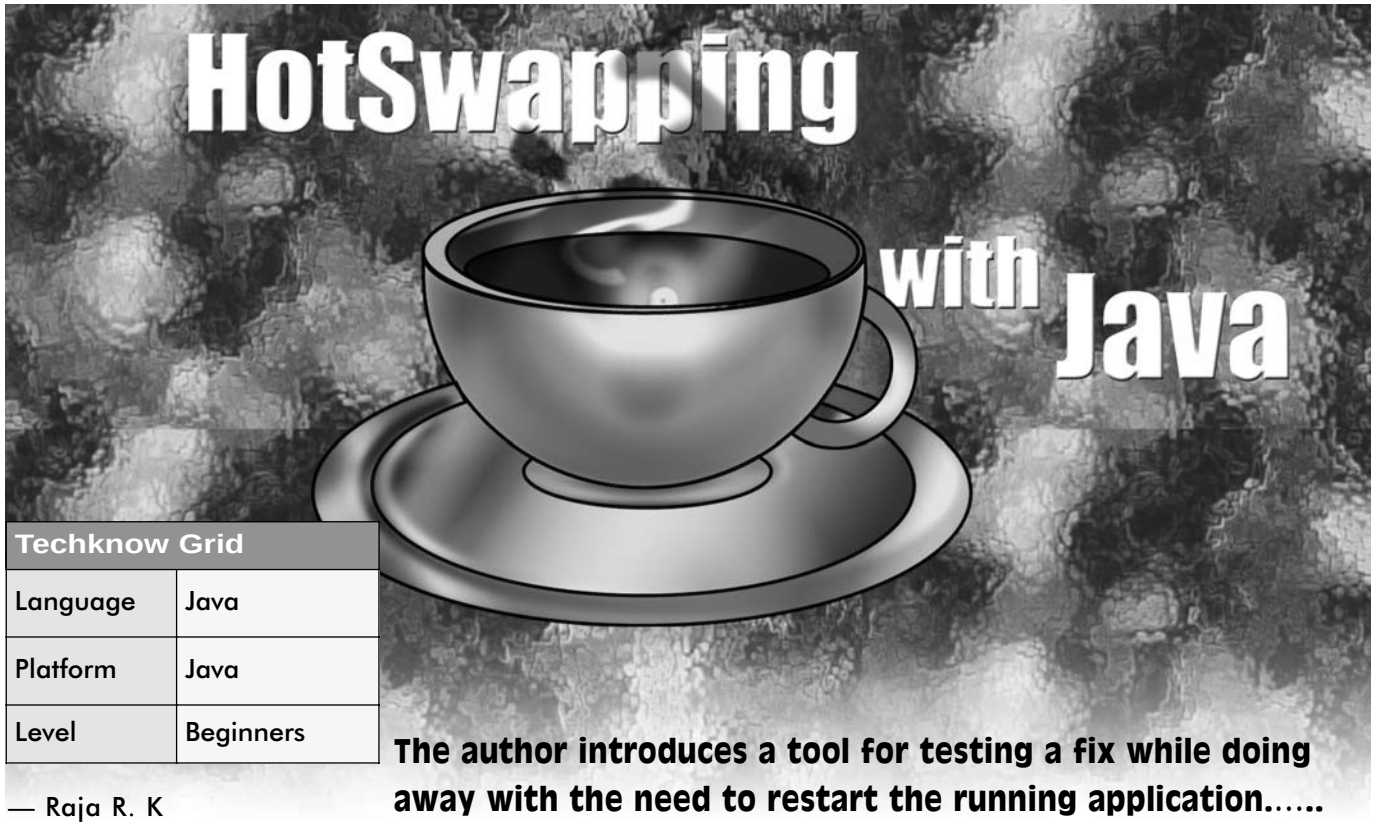
HotSwapping with Java



Techknow Grid	
Language	Java
Platform	Java
Level	Beginners

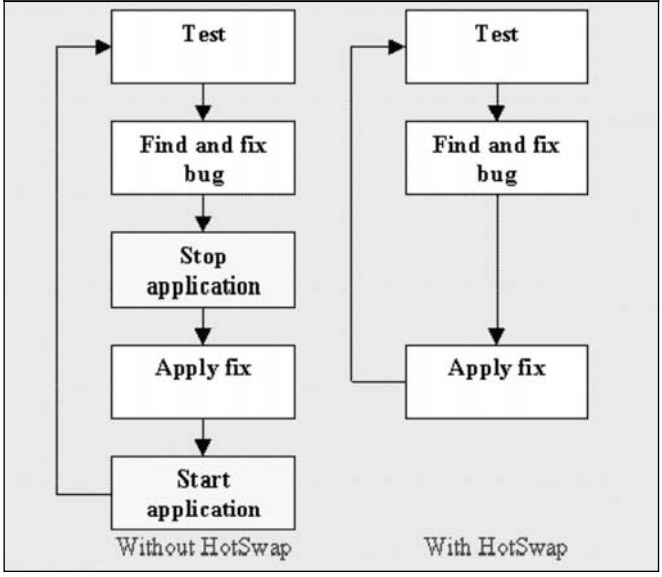
The author introduces a tool for testing a fix while doing away with the need to restart the running application.....

— Raja R. K

Introduction:

There may be occasions when you find bugs in a running application, fix it and want to test the fix without restarting the application. This is very common if you are working on enterprise level applications, which have less or absolutely no acceptable downtime. In this article we will show you how one could utilize the Java HotSwap technology in such scenarios.

Fig 1 **Java HotSwap Technology Advantage**



The Tool:

HotSwap adds ability to the Java platform (starting Java 1.4) for replacing Java classes on the fly. This dynamic class redefinition helps the developer to substitute an old class instance with a new one without the need to restart the running application.

Figure 1 illustrates the advantages of Java HotSwap technology.

If you want to test your fix without using HotSwap technology, you have to restart the running application after applying the fix. This is definitely a time consuming and annoying process.

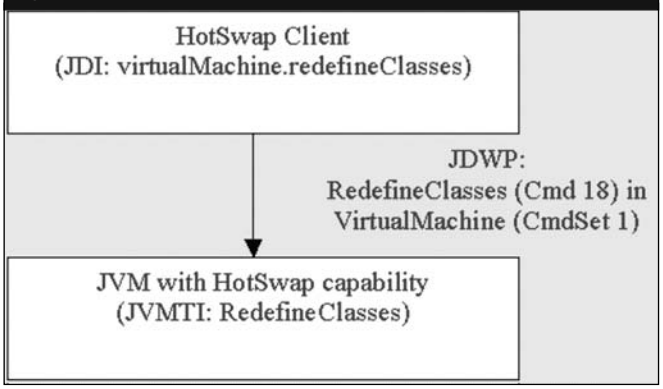
HotSwap technology provides the following benefits to developers:

- Facilitates on-fly patching for applications that cannot be interrupted;
- Facilitates on-fly debugging by allowing the developer to add more print statements to the running application; and
- Profiling (NetBeans JFluid uses the Code HotSwapping technology).

Java HotSwap technology is based on the Java Platform Debugger Architecture (JPDA). The following picture (**figure 2**) shows a higher level architecture view of Java HotSwap technology:

HotSwap client tool implements Java Debugger Interface (JDI). Lots of free client tools are available on the markets that support HotSwap functionality, including IBM's Eclipse and Sun's HotSwap Client Tool.

Fig 2 Java HotSwap Architecture



The HotSwap client tool communicates with local or remote HotSwap-capable JVM using Java Debugger Wire Protocol.
The power and usefulness of Java HotSwap technology could be best explained with a simple example.
Consider the following Java program (**code 1**):

Code 1

```
public class Simple
{
    public void display()
    {
        System.out.println("power of java hotswap
technology!!!");
    }

    public static void main(String args[])
    {
        Simple s = new Simple();

        while(true) {
            s.display();

            try {
                Thread.sleep(1000);
            } catch(Throwable ignore) {
            }
        }
    }
}
```

The above program just displays the message 'power of java hotswap technology' in an infinite 'while' loop with a sleep interval of 1,000 milliseconds.

We will modify this program on-the-fly using Java HotSwap technology so that it displays the string in the following format: Power of Java Hotspot Technology. We will use Sun's HotSwap client tool for dynamic class redefinition. See reference section for download location.

Start the original program in debug mode as shown below:

```
java-hotspot
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=
4000 -Xdebug -Xnoagent -cp . Simple
```

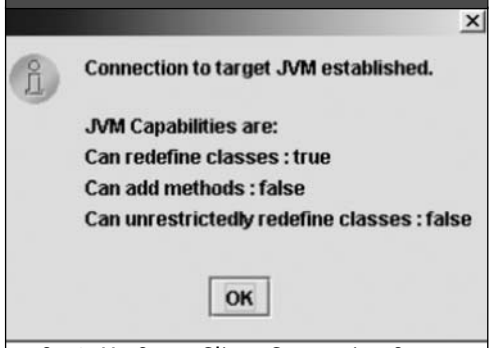
Launch Sun's HotSwap client and configure the old and new classpaths and source path locations. Next, configure the connection settings as show in **figure 3**.

Fig 3

Sun's HotSwap Client Connection Settinas Window

Connect Sun's HotSwap client to our application by selecting JVM > Connect menu option. If connected successfully, you should see a popup window similar to the one shown in **figure 4.**

Fig 4



Sun's HotSwap Client Connection Status

The popup window describes the remote JVM capability for dynamic class redefinition. Once connected, you can redefine the old class by loading the modified class either by selecting File > Open New Class File or JVM > Find Changed Classes menu option.

Modify and compile the program to display the message in the format Power of Java Hotspot Technology.
Selecting JVM > Find Changed Classes in the Sun's HotSwap client tool will show the list of modified classes. Refer **figure 5**.

From the above figure you can see that the only difference

Fig 5 Sun's HotSwap Client Tool

Fig 6 Dynamic Class Redefinition using Java HotSwap Technology

```
D:\rajkanna\articles\developer_iq\HotSwap\example>D:\java_1.4.1_02\bin\java -
hotspot -Xrunjdwp:trans
port=dt_socket,server=y,suspend=n,address=4000 -Xdebug -Xnoagent -cp . Simple
power of java hotswap technology!!!
power of java hotswap technology!!!
power of java hotswap technology!!!
Power of Java Hotswap Technology!!!
Power of Java Hotswap Technology!!!
Power of Java Hotswap Technology!!!
```

between our old and new program is the string format. To submit the modified class, click on the Submit button. Once submitted, you can see that the changes we made to the modified class are reflected in our running application as shown in **figure 6**.

The following steps will demonstrate how you can achieve the same result using Eclipse.

1. Create an Eclipse project the new program.
2. Build and run the project in debug mode, as shown in **figures 7** and **8**.

Fig 7 Eclipse Debug Launch Window

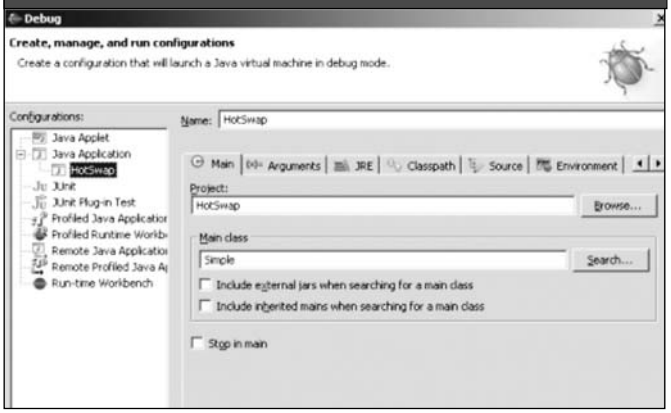
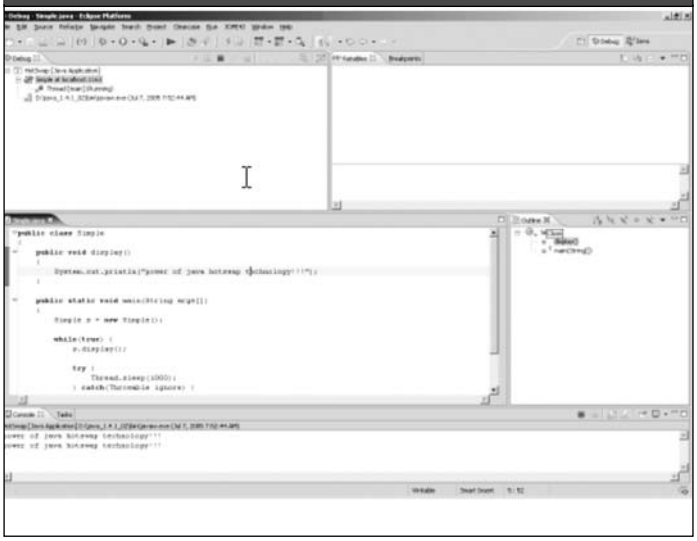


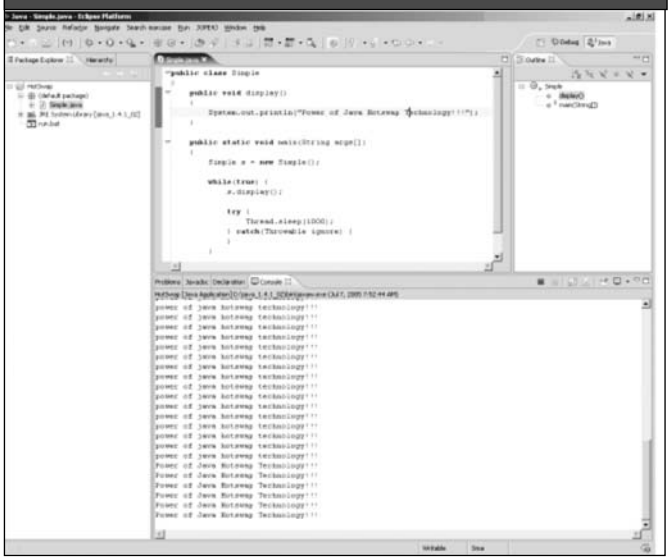
Fig 8 Eclipse Debug Perspective



3. While running in debug mode, change the string format and compile the project for Eclipse to automatically perform HotSwap for you, as shown in **figure 9**.

Please remember that we did all this without restarting the already running application. Really great isn't it?

Fig 9 Automatic HotSwapping with Eclipse



Reference

- http://www.java.sun.com
- http://www.java.sun.com/j2se/1.4.2/docs/guide/jpda/
- http://www.experimentalstuff.com/Technologies/HotSwapTool/index.html

About the author

Raja R.K is a Lead Engineer working at HCL Technologies (Networking Product Division), Chennai. He has more than six years of experience in Software Development. He can be reached via email at: rajark_hcl@yahoo.co.in